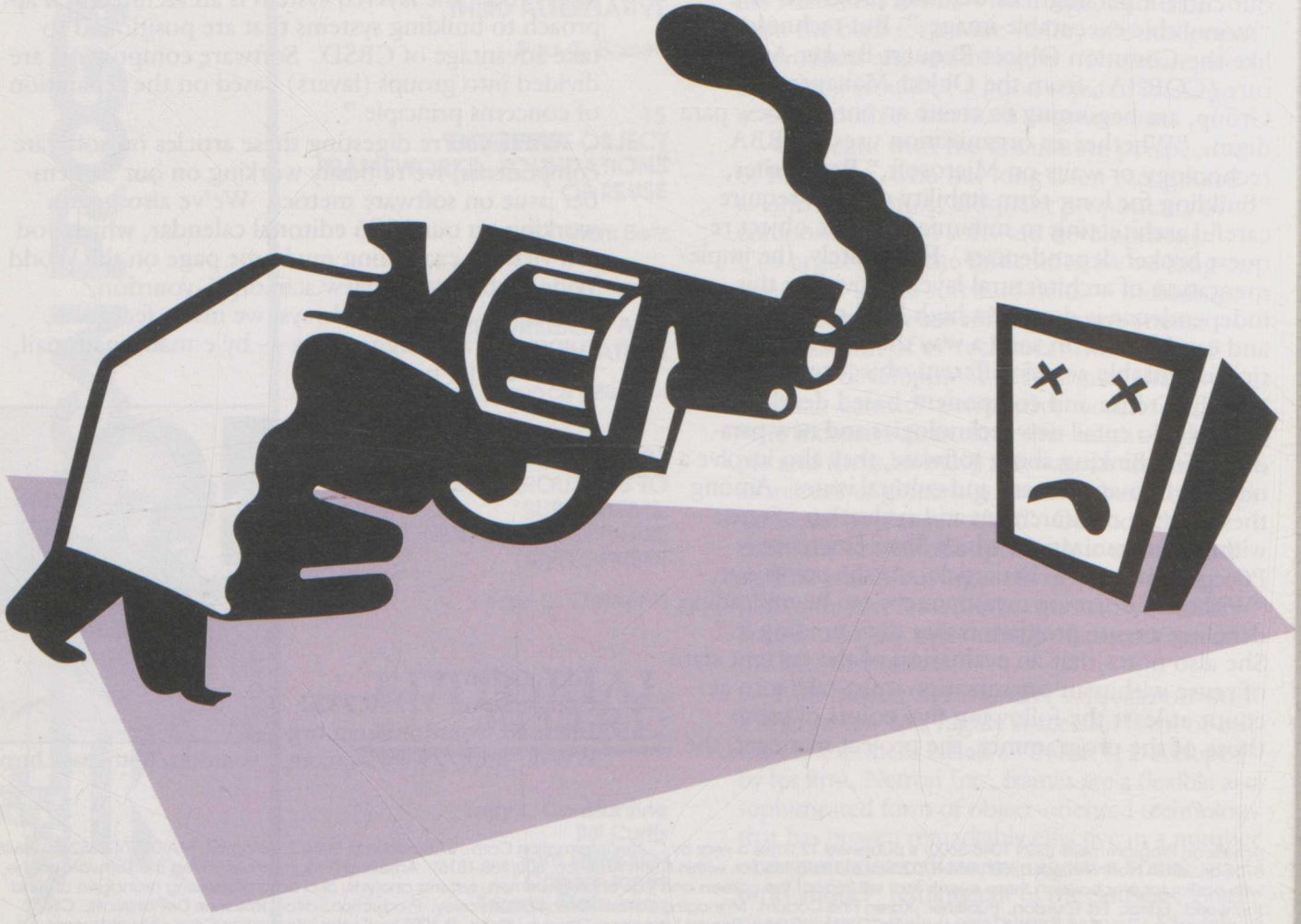
"No Silver Bullet" Reconsidered

By Brad Cox



Denver's new international airport was to be the pride of the Rockies, a wonder of modern engineering. . . . Even more impressive than its girth is the airport's subterranean baggage-handling system. Tearing like intelligent coal-mine cars along 21 miles of steel track, 4,000 independent "telecars" route and deliver luggage between the counters, gates, and claim areas of 20 different airlines. A central nervous system of some 100 computers networked to one another and to 5,000 electric eyes, 400 radio receivers, and 56 bar-code scanners orchestrates the safe and timely arrival of every valise and ski bag.

At least that is the plan. For nine months, this Gulliver has been held captive by Lilliputians — errors in the software that controls its automated baggage system. Scheduled for takeoff by last Halloween, the airport's grand opening was postponed until December to allow BAE Automated Systems time to flush the gremlins out of its $193-million system. December yielded to March. March slipped to May. In June the airport's planners . . . conceded that they could not predict when the baggage system would stabilize enough for the airport to open.

— W. Wayt Gibbs,
Scientific American [5]

CRISES VERSUS SILVER BULLETS

The software crisis has been with us now for nearly a quarter-century. During this time, the computer industry has progressed at breakneck speed through the computer revolution, the personal computer revolution, and most recently, the network revolution triggered and/or accelerated by the explosive spread of the Internet and the World Wide Web. Yet during the very period that the computer industry has been delivering exponential improvements in price-performance, software remains intractable, immune to the organizational innovations that make us think of computer manufacturing as a mature industrial-age enterprise.

In one of the most influential books of this era, *The Mythical Man-Month*,[1] Fred Brooks observes that adding more people to a late software project only seems to make matters worse [2]. And in an equally influential article, "No Silver Bullet: Essence and Accidents of Software Engineering," he argues that the difficulties are inevitable, arising from software's inescapable *essence*, as distinct from *accident*, or something we're doing wrong in producing it [3].

This article will reexamine Brooks's argument from a different perspective to arrive at precisely opposite conclusions. Whereas Brooks adopted the technocentric perspective of the software industry's established paradigm, we'll step outside of this frame to examine it from a human-centric viewpoint. We'll not focus our microscope on how we build software today. We'll focus instead on the people, or more precisely, on the incentive structures that might lead people and firms to build software in the future as mature engineering domains build baggage carts and airplanes today.

This difference in viewpoint leads to an entirely different interpretation of the same bundle of facts that Brooks considered. The good news is that a silver bullet is indeed feasible for whoever musters the determination to forge it. The bad news is that, exactly as with the silver bullets that eliminated other crises throughout history, our silver bullet will also be a paradigm shift, not a technology. Paradigm shifts can be very bad news, for they challenge the most deeply held beliefs of established incumbents.

Indeed, incumbents who resist new paradigms to the end often find themselves displaced by newcomers who don't share the same system of beliefs. The freedom-loving Indians of the American frontier were once exactly such incumbents. They were emphatically displaced by property-conscious European invaders during the confrontation that the winners called the Taming of the Wild West.

NO SILVER BULLET?

Let us consider the inherent properties of this irreducible essence of modern software systems: complexity, conformity, changeability, and invisibility.

— Fred Brooks [3]

As Brooks suggests, "The complexity of software is an essential property, not an accidental one." We observe that this inherent complexity derives from four elements: the complexity of the

---

This article has been adapted by the editors from the forthcoming book *Superdistribution: Objects as Property on the Electronic Frontier* by Brad Cox (Reading, MA: Addison-Wesley, 1995). © 1995 by Addison-Wesley Publishing Co.

[1]Earlier this year, Addison-Wesley published a 20th anniversary edition of this software enfineering classic. The new edition contains four new chapters. See Frederick P. Brooks, The Mythical Man-Month, anniversary ed. (Reading, MA: Addison-Wesley, 1995).

problem domain, the difficulty of managing the development process, the flexibility possible through software, and the problems of characterizing the behavior of discrete systems.

— Grady Booch [1]

Notice that something is missing from each of these lists. The four "essential" properties in these quotations are all symptoms, not causes. They all originate from a causal difference that is so deep, yet so obvious, that neither author even mentioned it in his list. This underlying difference is that whereas tangible goods are made of atoms, electronic goods are made of bits. I will argue here that this difference produces not only the symptoms that these authors listed, but it also explains why the Denver airport's tangible goods — escalators, baggage carts, telephones, and so forth — are in place and functional, while the software has been a dismal failure. In other words, I'll argue that the underlying cause of the software crisis is a breakdown in human incentive structures that originates from the problems of buying, selling, and owning goods made of bits.

At first, this difference may seem so obvious that little good comes from mentioning it. But I will show here that it holds the key insight from which a silver bullet might someday be forged. Since tangible goods abide by conservation of mass, it is hard to replicate and transport them but trivial to buy, sell, and own them. Therefore large commercial enterprises have arisen to provide them. These enterprises are capable of cooperating across vast differences in space, time, language, and culture to master the vast complexity of even the most mundane of tangible goods. In the Denver International Airport example, these goods include not only the conveyor belts, airplanes, and buildings at the top of the manufacturing pyramid, but also the subcomponents used in building them, right down to the raw materials from the mines, wells, and farms.

The situation with electronic goods is exactly the reverse. Electronic goods can be replicated and transported at the speed of light, but it is hard to buy, sell, and own them. Since bits do not abide by the physical laws upon which economics has been based since antiquity, the software industry has been unable to mobilize a similar structure of production for the comparable subcomponents we call "reusable software."

This rediagnosis of the causal factor leads to a solution to the software crisis that, following Brooks, I've called the silver bullet. If we could deploy a way of buying, selling, and owning goods made of bits comparable to the incentive structure of tangible manufacturing — a system that would work not only for the large objects at the top of this pyramid, but also for multigranular subcomponents as they flow through a comparably elaborated software structure of production — we'd have a solution exactly like the one that successfully produced the tangible goods for the Denver Airport.

This article will conclude by outlining how this could work if we should ever muster the determination to build the infrastructure to support it. The good news is that the technical difficulty of building such an infrastructure is quite small, well within the capabilities of today's computer industry. The bad news is that, exactly as with all the other silver bullets in human history, the real innovation doesn't lie in the infrastructure technology itself. The innovation is a paradigm shift, a redefinition of what it actually means to say we buy, sell, or own objects made of bits.

MANUFACTURING'S SILVER BULLET

What is more complicated, the mundane wooden pencil a Denver ticket agent might use to write out your ticket or its electronic equivalent, a word processor? If you answered the word processor, you're in good company. Like Brooks and Booch, programmers invariably agree that developing computer software is the most complicated of all human activities. I've heard people seriously compare word processors with the complexity of Boeing's airplanes.

But according to the product manager of Microsoft Word (version 3.1), his software development team consisted of only eight programmers. (This figure includes only those doing hands-on work with the code, not those who documented, tested, and marketed it.) His numbers are comparable to my own experience with software development team sizes, although perhaps not your own. Use your own estimate of the team size for your own last project, then contrast the size of your team with the one that worked together to put those mundane wooden pencils in your desk drawer.

I, Pencil

I, Pencil, simple though I appear to be, merit your wonder and awe, a claim I shall attempt to prove. . . . I have a profound lesson to teach. And I can teach this lesson better than can an automobile or an airplane or a mechanical dishwasher because — well, because I am seemingly so simple.

Simple? Yet, not a single person on the face of this earth knows how to make me. This sounds fantastic, doesn't it? Especially when it is realized that there are about one and one-half billion of my kind produced in the U.S.A. each year.

Pick me up and look me over. What do you see? Not much meets the eye — there's some wood, lacquer, the printed labeling, graphite lead, a bit of metal, and an eraser. Just as you cannot trace your family tree back very far, so is it impossible for me to name and explain all my antecedents. But I would like to suggest enough of them to impress upon you the richness and complexity of my background.

My family tree begins with what in fact is a tree, a cedar of straight grain that grows in Northern California and Oregon. Now contemplate all the saws and trucks and rope and the countless other gear used in harvesting and carting the cedar logs to the railroad siding. Think of all the persons and the numberless skills that went into their fabrication: the mining of ore, the making of steel, and its refinement into saws, axes, motors; the growing of hemp and bringing it through all the states to heavy and strong rope. . . .

The logs are shipped to a mill in San Leandro, California. Can you imagine the individuals who make flatcars and rails and railroad engines and who construct and install the communication systems incidental thereto? . . .

Consider the millwork in San Leandro. The cedar logs are cut into small, pencil-length slats less than one-fourth of an inch in thickness. These are kiln dried and then tinted. . . . The slats are waxed and kiln dried again. How many skills went into the making of the tint and the kilns, into supplying the heat, the light and power, the belts, motors, and all the other things a mill requires? . . .

My "lead" itself — it contains no lead at all — is complex. The graphite is mined in Ceylon. Consider these miners and those who make their many tools and the makers of the paper sacks in which the graphite is shipped and those who make the string that ties the sacks and those who put them aboard ships and those who make the ships. . . .

The graphite is mixed with clay from Mississippi in which ammonium hydroxide is used in the refining process. Then wetting agents are added, such as sulfonated tallow — animal fats chemically reacted with sulfuric acid. After passing through numerous machines, the mixture finally appears as endless extrusions — as from a sausage grinder — cut to size, dried, and baked for several hours at 1,850 degrees Fahrenheit. To increase their strength and smoothness, the leads are then treated with a hot mixture which includes candelilla wax from Mexico, paraffin wax, and hydrogenated natural fats.

My cedar receives six coats of lacquer. Do you know all of the ingredients of lacquer? Who would think that the growers of castor beans and the refiners of castor oil are a part of it? They are. Why, even the processes by which the lacquer is made a beautiful yellow involves the skills of more persons than one can enumerate! [6]

Managing Complexity

This tale of the wooden pencil brings a crucial but slippery distinction into the software complexity debate. Although it might be academically amusing to compare the complexity of wooden and electronic pencils in absolute terms, arguing about absolute complexity doesn't really move the ball ahead. What does is realizing that encapsulated complexity is no longer complexity at all. It's gone, buried forever in somebody else's problem. This shifts the focus to the human systems that manage complexity successfully versus those that don't. The productive question is, "How do wooden-pencil makers encapsulate complexity so successfully that the rest of us have the luxury of forgetting that the complexity is even there?"

They do this by ensuring that somebody else has an incentive to hide the complexity for us. The quantum event that underlies their entire system is the commercial exchange transaction. This gives the office supply store its incentive to hide from us the complexity of producing our own wooden pencils. The same principle extends downward through the structure of production pyramid that supports that store. It persuades the San Leandro mill owner to remain in the business of hiding complexity from the office supply store. It persuades the lumberjack to remain in the business of hiding the complexity of logging from the mill. And it persuades the rapeseed farmer in the East Indies to remain in the business of hiding the complexity of his farm from everyone else in this global production chain.

It really doesn't matter whether wooden pencils are more or less complex than electronic pencils in absolute terms. What really matters is that, for wooden pencils, a thriving human system operates to hide complexity from

everyone else in the chain. This system does its job so well that the rest of us have the luxury of entirely forgetting that it was ever even there. And if that doesn't qualify as a silver bullet for their complexity problems, I'd have to agree with Brooks that there really is no hope for us.

PENCILS AS THE FRUIT OF THE PENCIL TREE

This line of argument helps us to see that complexity is not the fundamental cause of the software crisis. Complexity is itself a symptom of a much deeper disease. In fact, this disease is at the deepest level imaginable, so deep that it didn't even make it onto Brooks's and Booch's list of software's essences.

The disease is a breakdown in the very commercial exchange transaction upon which tangible manufacturing's complexity-hiding mechanism is based. The path out of the software crisis involves appreciating how their solution actually works and then designing one for software that has exactly the same human properties. The trick is that ours cannot rely on the conservation-of-mass laws that support commerce in tangible goods made of atoms. Our system needs a different fundamental basis, one that works for goods that do not obey the physical conservation laws upon which the traditional commercial exchange transaction relies.

The important thing to notice here is that conservation of mass has reach. It applies not only to the wooden pencils, but with equal force to every one of the subcomponents used in building them. It is as if the pencil in every customer's hand remains connected to the vast global tree of human relationships that produced it. Conservation of mass ensures that each trace of graphite a pencil leaves on paper automatically draws the right amount of production from each of the lower levels of the tree.

In this way, the consumption of a pencil in the Denver Airport is infallibly communicated, through price signaling, right down to the bottom levels of the tree. Price signalling tells the graphite miner in Ceylon how much graphite should be mined. It tells the mill how much demand exists so it will produce the right number of pencils to satisfy that demand. The pencil tree is an ecological system, exactly like a peach tree, in that it draws the right amount of water and sunshine from nature and transforms them into the right number and amount of amino acids, sugars, and proteins. All of these are precisely coordinated to support the production of fruit that we think of as the tree's commercially significant product.

In other words, the system that's operating here is recursive in a particularly powerful way. It doesn't just work at the level of peaches, pencils, and word processors — the levels the world thinks of as commercially significant. It doesn't apply only to the peaches, but with precisely equal force down through the leaves, the twigs, the branches, the trunk, and the roots.

In contrast, since electronic goods don't abide by conservation of mass, the vessels inside our electronic pencil tree get vapor lock. So the tree dies. Or more precisely, such trees never got the chance to evolve in the first place.

Demand for computer software is high, so high prices encourage us to provide large-granularity computer software such as word processors. But the smaller-granularity objects we might use to build them can be replicated — without yielding further revenue to their owner — each time an application that relies on them is sold. Thus there is no incentive for people to populate the lower levels of the tree. Since the demand pulse doesn't propagate recursively through the tree, we all naturally gravitate to the levels where it is possible to get paid. We all work on building large applications, and the structure of production trees never evolve. Since supporting levels don't evolve, the entire task falls to the team at the very top of what would otherwise be a cooperative human pyramid. And we all gather at software engineering conferences to complain of the crushing complexity of our task.

OUT OF THE CRISIS

Unlike the hardware industry, which has organized itself into a fully elaborated rainforest of mutually interdependent structure of production trees, the software industry remains stuck in the unicellular, bacterial stage of the primordial ooze. I'll avoid here the question of whether programmers might be individually better off if our industry remained in the primordial ooze stage. My book engages that question in detail by examining how the analogous confrontation between "primitive" and "advanced"

social orders developed in the American West [4]. Here I'll simply take this industry's professed interest in "doing something" about the software crisis at face value and concentrate on showing what would be needed to evolve an advanced structure of production tree for goods made exclusively of bits.

Notice that this cannot be addressed solely at the level of large-granularity computer applications. It involves paying attention to how we can grow healthy supporting layers — healthy roots, trunks, and branches — not just commercially significant fruit. As we've seen, the fruit is the only level that receives serious attention today.

Copy-protection technologies, including dongles, shareware-unlocking technologies, license servers, and encryption, are all unigranular solutions. They enforce ownership of objects that are large enough to stand on their own as commercial products, but they do nothing for smaller-granularity components from which large-granularity objects might someday be composed. Yet large applications are the one level of the tree that least needs greater protection. After all, applications can be protected by encasing them in paper and cellophane, at which point they can be bought and sold like cornflakes in an ordinary industrial-age software store.

The path out of the crisis involves finding a viable replacement for the quantum event at the core of the economic tree that produces tangible goods such as pencils; that is, the commercial exchange transaction. Most commonly, this event relies on conservation of mass to define exchange in terms of acquisition of atoms. But even in tangible domains, acquisition of atoms is not the only basis of owning. The path out of the software crisis involves another interpretation of owning, which is based on acquisition of use instead.

We don't generally think of these two meanings as separate, because most of our daily transactions bundle them together. For example, when I buy a burger at a fast-food restaurant, I'm actually buying an ownership bundle. I acquire ownership of the physical atoms from which the burger is composed. Conservation of mass immediately kicks in to guarantee all parties that the restaurant owns one less burger, or more precisely, the bun, meat, and condiments from which burgers are assembled. My acquisition of these atoms instantly generates a scarcity pulse that travels throughout a vast agrarian system, encouraging farmers and distributors around the globe to produce more.

But ownership of the atoms is not the only thing I buy with my burger. I've implicitly acquired as part of my rights bundle the right to use those atoms as I please. I can eat the burger myself, give it to a friend, feed it to my dog, or just throw it away. In electronic terms, my burger transaction involves not just copyright, which is enforced in this case by natural law. It also involves an exchange of useright, which is implicit and therefore rarely articulated as a separate right.

So, defining ownership in terms of acquisition of atoms is by no means the only way to buy and sell things — it is only the most common. The person who owns the atoms can, alternatively, retain the atoms on his or her balance sheet but sell the right to use those atoms to someone else. Hertz and Avis are masters at this business. They buy automobiles, which are recorded on the asset side of their ledger. They sell the right to use those atoms to customers by the day or the mile.

This is exactly where our escape from the software crisis could be launched. Today everyone is accustomed to buying software the way we buy burgers. We acquire the right to use software bundled with the bits. However, deep down, we also know that bits add absolutely no value to our balance sheet. Bits have never been an asset; in fact, they are really a liability. They occupy physical storage space that might be used for something else. And again, unlike atoms, they represent no cost to the provider, since they generate no scarcity pulse that requires their subcomponent providers to produce more.

So why don't we just give them away? After all, bits are just packaging, and it costs nothing to provide them. The real asset is the software utility that they encode. What people want is the right to use the utility encoded in the bits.

So why not just let people acquire the bits freely and base revenue collection on acquisition of use instead? This shift in what it means to buy, sell, and own immediately shifts the ownership issue onto ground where technology can apply. For although computer software is intrinsically

unable to monitor its acquisition, it is easily able to monitor its use. Consider the following example:

```
if (query()) {      /* is this a paid up customer? */
      ...           /* deliver requested service */
      commit(); /* record successful delivery */
} else {            /* refuse service */
      ...
}
```

These query and commit calls rely on communication infrastructure that doesn't exist today but that could be provided relatively easily. Since this infrastructure deals with money, it would have to be sufficiently tamper resistant to be trusted by all parties. An infrastructure capable of conveying the usage information captured by these query and commit calls to a financial institution for billing is "just a matter of engineering" and quite within the reach of today's computer industry.

The key to this approach is a paradigm shift in which buying, selling, and owning are based on acquisition of utility encoded in the bits instead of acquisition of the bits themselves. Invocations have exactly the same recursive properties for software that conservation of mass has for commerce in tangible goods. For example, when a large object runs on an end user's PC, the user will be charged for using this application. However, the application's owner will be charged for his or her application's use of any reusable subcomponents that it invokes. Just as conservation of mass conveys scarcity pulses throughout global industries, metering of invocation can carry energy right from the top to the roots of the structure of production tree.

REFERENCES

1 Booch, Grady. *Object-Oriented Design with Applications.* Redwood City, CA: Benjamin/Cummings, 1991.

2 Brooks, Frederick P. *The Mythical Man-Month.* Reading, MA: Addison-Wesley, 1975.

3 Brooks, Frederick P. "No Silver Bullet: Essence and Accidents of Software Engineering." *IEEE Computer*, Vol. 20, no. 4 (April 1987), pp. 10–19.

4 Cox, Brad. *Superdistribution: Objects as Property on the Electronic Frontier.* Reading, MA: Addison-Wesley, forthcoming 1995.

5 Gibbs, W. Wayt. "Software's Chronic Crisis." *Scientific American*, Vol. 271 (September 1994), pp. 86–95.

6 Read, Leonard. "I, Pencil." *The Freeman* (December 1958), pp. 32–37.

Brad Cox is a faculty member of the George Mason University Program on Social and Organizational Learning (PSOL). PSOL is a radically interdisciplinary department that concentrates on overcoming socioeconomic and technical obstacles to change, development, and learning as firms transition to a global information-intensive economy. His educational interests are in applying Internet, television, and groupware technology to expedite experiential and collaborative learning experiences in the academy, the home, the workplace, and the world.

Dr. Cox is the author of Object-Oriented Programming: An Evolutionary Approach, *a book that is generally credited with launching today's industry-wide enthusiasm for object technology. He cofounded the Stepstone Corporation and originated the Objective-C programming language and Software-IC libraries in order to launch a software industrial revolution where software would be produced, not by fabricating everything from first principles, but by assembling interchangeable (reusable) software components from a market in interchangeable components. Dr. Cox recently founded the Coalition for Electronic Markets, the objective of which is to build and deploy a nationwide revenue collection infrastructure for commerce in electronic goods.*

Dr. Cox received his Ph.D. from the University of Chicago for theoretical and experimental work in neurophysiology in an area that has since become known as neural networks. His postgraduate experimental studies were at the National Institutes of Health and at the Woods Hole Marine Biological Laboratories.

Dr. Cox can be reached at George Mason University, 4400 University Drive, Fairfax, VA 22030-4444 (703/968-8229; Internet: bcox@gmu.edu; WWW: http://web.gmu.edu/bcox). ★